AD696934

CONTENTS

Abstract	ii
Problem Status	ii
Authorization	ii
INTRODUCTION	1
TESTING COMBINATIONAL LOGIC	1
Generation of Test Dictionary	2
Application of Test Dictionary	5
Bounds on the Number of Tests	5
Serial Vs Fixed Test Schedules	6
TESTING SEQUENTIAL MACHINES	6
Philosophy of Diagnosing Sequential Machines	7
State Table Methods	7
Search Method	8
SUMMARY	12
FUTURE WORK	12
REFERENCES	13

ABSTRACT

Finding efficient testing procedures for digital machines has become of prime concern to the industry due to high circuit densities brought about by evolution in semiconductor device technology.

The number of inputs required for testing can be reduced from the set of all possible input combinations to a smaller group derived from a dictionary of tests which defines unique relationships between failure mode and output response. Reduction of the test dictionary by prime-implicant minimization methods and by serial application of the reduced set can lead to manageable test programs. These involve sequence lengths characterized by a lower limit, bounded by the logarithm of the number of initial table entries.

Testing sequential machines is especially cumbersome due to the dependency of circuit response on previous input sequences. Lines of attack include checking for consistency between circuit response and design state table. A more manageable scheme involves grouping machines of specific failures according to their responses to selected test inputs. A proper choice of inputs for each test step and the adaptive application of the resulting sequences provides relatively efficient fault-location programs.

Practical schemes for complete testing have yet to be found. Adaptive sequence generators with programmed options for state-table and combinational analysis appear to yield more promising results.

PROBLEM STATUS

This is an interim report on a continuing problem.

AUTHORIZATION

NRL Problem B01-04
Project RR003-02-41-6150

Manuscript submitted May 26, 1969.

DIGITAL DIAGNOSTIC TECHNIQUES: SURVEY AND RECOMMENDATIONS

INTRODUCTION

The development of integrated circuitry and the mass production of large-scale digital hardware have posed new problems for the industry. Among them is how to check the operation of digital machines without expending excessive man hours of labor. One obvious approach is automation, and it is now being extensively used by most large suppliers of electronic devices. Yet, there are still problems which relate to the specific algorithms used to generate the set of tests to apply to the device in question. The particular choice of test-generating algorithms is critical in that the number of input combinations to be searched, for all but the smallest machines, can become extremely large for complete testing if, indeed, complete testing is possible. This point is especially relevant when only a few machines of a design are being produced.

It should be noted that, with the advent of integrated circuitry, modular construction, and microscopic dimensions, the traditional probing of internal circuitry is generally not feasible, and tests have to depend on the input-output relationships of the machine as a black box. Test points can be brought out from strategic circuit areas, but this practice thwarts the ideal of designs with low pin-to-gate ratios.

Digital systems contain both combinatorial and sequential logic which have to be tested. The problem of generating tests for combinatorial logic has been approached, with feasible results being obtained by Kautz (1) and by Hornbuckle and Spann (2). However, a complete solution for the general class of sequential machines has not been arrived at, although powerful techniques for reasonably thorough checkout have been devised.

TESTING COMBINATIONAL LOGIC

In combinational logic, the response to each test input is independent of the previous input sequence. Therefore, an exhaustive logic test sequence for a circuit having n inputs consists of 2^n tests. A method of reducing the required number of tests is to list a set of faults to be considered and then to design tests to detect the occurrence of these faults. Certain assumptions are generally made about these faults. First, they are considered not to be intermittent; i.e., once a fault has occurred it remains long enough to be detected and corrected. Second, while there are methods for detecting multiple failures, only single faults are considered, multiple faults having small occurrence probabilities; also, it is intuitively logical that most tests derived for detecting single faults will also indicate abnormalities when additional failures are present in the system. Third, the types of fault generally considered are those which cause any input or output line within the system to be stuck in a 1 or 0 condition. Less often, open connections or shorts between different lines are taken into consideration.

The following is a discussion of the aspects of specific techniques applicable to combinatorial machines. It is assumed that a model on which the machine in question and all designated faults can be simulated is available. Most likely, this would be a general-purpose computer programmed as a logic simulator.

Generation of Test Dictionary

The first step in the test-generation procedure is the formation of a fault-test dictionary. This dictionary is a table which lists, for each input combination, the associated failure conditions causing abnormal output combinations. From here on, the digital unit operating error-free will be referred to as the good machine m_0, and the unit under a specific failure condition i as the bad machine m_i The central problem then, is to find the complete set of inputs and associated bad machines m_i for which the output of the good machine m_0 and the m_i differ.

Two approaches have been used in constructing a fault-test dictionary:

1. Entries in the fault dictionary can be generated by starting with the input list and finding the set of faults corresponding to abnormal outputs for each input combination;

2. Conversely, given any fault, one can find that set of inputs which causes the output to be different for the good machine and the specified faulty machine.

The first method of finding fault-test entries is conceptually straightforward.

Let us assume that the logic system is fully described in a functional block simulation. Then starting with input $\bar{x}_1, \bar{x}_2, \bar{x}_3, \ldots, \bar{x}_n$ through all combinations to $x_1, x_2, x_3, \ldots, x_n$, each failure mode of interest can be simulated and the output noted in each case.

After all input combinations have been applied for each simulated failure mode, a table can be set up as below.

x_n	...	x_2	x_1	f_0	f_1	f_2	f_3	...	f_i
0		0	0	0	1	0	0		0
0		0	1	1	1	0	1		1
0		1	0	0	0	0	1		1
...		...	...	...	...	...	...		...
1		1	1	0	1	1	0		1

The f_i's are the output of the machine for each simulated failure i and input $x_1, x_2, x_3, \ldots, x_n$. Note that no assumption is made about the nature of the faults, whether they are single, multiple, short circuits or open circuits.

It is clear that generating the fault dictionary by input simulation requires $2^N F$ complete simulations of the machine, where N is the number of input lines and F the number of failure modes.

The second technique is the failure sensitivity method, whereby the number of operations involved in setting up the test dictionary can be reduced. The approach is to assume a failure and then, using the machine description, generate that set of inputs which sensitizes the machine output to each failure listed.

In this way it should be feasible to generate the test dictionary with only F sensitivity calculations. Whether the advantages of the smaller number F of sensitivity calculations is offset by the greater complexity of each calculation, as compared with direct simulation, is another problem.

Two sensitivity methods have been presented in the open literature.

1. The first method uses an operation which has been called the "Boolean difference" by Sellers, Hsiao, and Bearnson (3). Let the good machine be denoted by its function $F(x)$ and the machine under failure by $F^B(x)$. Find the input states under which $F(x) \neq F^B(x)$. This is the condition under which

$$F(x) \oplus F^B(x) = 1 ,$$

or

$$F(x) \cdot \overline{F^B(x)} + \overline{F(x)} \cdot F^B(x) = 1 .$$

For a single "stuck at 1 or 0" type of failure in terminal i, then

$$D_B\, F(x_i) = F(x_i, x_2, \ldots, x_i, \ldots, x_n) \oplus F(x_1, x_2, \ldots, \overline{x}_i, \ldots, x_n) ,$$

where D_B is the difference operator. Sellers and others (3) and Amar and Condulmari (4) have derived relationships which facilitate the application of the difference operator to the various Boolean forms. The enticing aspect of this method is that it uniquely defines those inputs and the states they must be in to sensitize the output to the state of terminal i.

It only remains to enter these input combinations including "don't cares" into their proper rows of the fault-test dictionary.

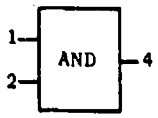
The disadvantages of this method are that it does not map the specific state of terminal i to the specific state of the machine output, and that memory requirements can become excessive when computing the Boolean difference for a function of many variables.

2. Another sensitivity method, called the D-Algorithm, after Roth (5,6) uses the internal line structure of a circuit to describe the function and its failures. In this method, each logical unit in the circuit is represented by a type of cubical complex known as a D-cube. It provides a representation of the sensitivity state between various terminals of a logic block and is derived from the singular cover as follows.

An AND gate with its singular cover and D-cube is shown in Fig. 1. The singular cover of a logic unit is a table that lists its terminal states for different outputs and completely describes its operation. The associated D-cubes can then be derived by matching all of the singular cover rows by pairs. Whenever a condition is found where the output and some input terminal differ in state, a new cover row is written with D's inserted for those terminals changing state. One can see, then, that D-cubes relate sensitivity states of pairs of logic unit terminals. Given a circuit made up of a number of interconnected logic units, the D-cubes for each unit are then linked so that terminal assignments are consistent within columns. This forms an array in which each column represents different terminals in the circuit and each row a different D-cube.

Now, finding a path through the circuit relating the output state to the state of some internal terminal involves finding a path through the D-cube array such that every D-cube element chosen is consistent in terminal state assignment with every other element. If a consistent sensitivity path is possible for some input condition, the algorithm will find it.

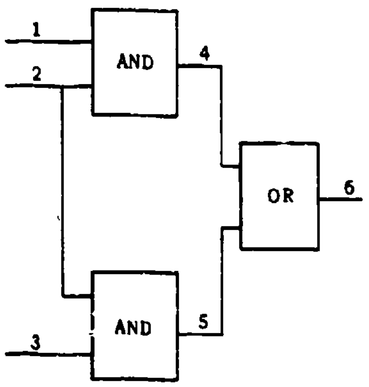
As a simple illustration of this method, the following procedure finds an input combination sensitizing the output, line 6 of Fig. 2, to the state of line 4, the AND gate block. The singular covers and associated D-cube table are shown with the circuit. Since we wish to have the circuit respond to state changes in line 4, first locate a D-cube in the



Terminals

	1	2	4
	1	1	1
	0	X	0
	X	0	0
D-cube	*D*	1	*D*
	1	*D*	*D*

Fig. 1 - Two-input AND gate with associated singular cover and *D*-cube



Terminals

1	2	3	4	5	6
1	1		1		
0			0		
	0		0		
	1	1		1	
	0			0	
		0		0	
			1		1
				1	1
			0	0	0
D	1		*D*		
1	*D*		*D*		
	D	1		*D*	
	1	*D*		*D*	
			D	0	*D*
			0	*D*	*D*
			D	*D*	*D*

Fig. 2 - Logic circuit with singular covers and *D*-cube

table with a D in column 4 (representing line 4). Picking $D^1 1^2 D^4$ specifies that a 1 on line 2 constrains the state of line 4 to follow that of line 1. Since line 2 is fixed at truth value 1, consistency demands that any next D-cube we choose from the table have a 1 or D in column 2. Before we do that however, let us continue finding a D-cube relating sensitivity in line 4 to states further along in the circuit. Picking $D^4 0^5 D^6$ satisfies this, specifying that the output, line 6, is sensitized directly to the state of line 4 when a 0 is on line 5. Combining the two D-cubes chosen thus far, we have $D^1 1^2 D^4 0^5 D^6$. Input line 3 has yet to be examined. Seeing that line 2 is in state 1 we could not, for instance, pick $D^2 1^3 D^5$, since 2 and 5 would be required to be in identical states, and this would be inconsistent with the assignments already made. Picking $1^2 D^3 D^5$ satisfies consistency and specifies a 0 on line 3, matching line 5. Combining the three D-cubes, we get $D^1 1^2 0^3 D^4 0^5 D^6$. We now have an input specified which sensitizes the output to the state of line 4 and have an input line, line 1, which can be used to exercise the logic for the test.

Application of Test Dictionary

Having derived a test dictionary as described above for a given machine, how can it be used to isolate a given fault or determine if the machine is good? Since we are concerned with distinguishing between columns F_i representing different machines, $(m_0, m_1, m_2, \ldots, m_i)$, we generate a new fault matrix G by the following method. For fault detection, where we are not concerned with distinguishing between various types of failures, we define a reference column, namely F_0, for the good machine. Then, in every fault column F_i corresponding to machine m_i where the machine output differs from that of m_0 we place a 1 in the appropriate row and column of matrix G. If the two outputs are identical, there is no distinguishability and we enter a 0. It is clear that the G matrix will have one less column than the F matrix.

It is appropriate at this point to bring in the notion of fault *diagnosis* as opposed to simple fault *detection*, if the reader has not already considered it. In fault detection, we are concerned only about whether the machine under test is working or not. In diagnosis, if the machine has failed, we wish to ascertain which of the faults listed is present. Fault diagnosis is also referred to as fault location.

When the G matrix is extended to fault diagnosis, then the reference columns become those faults which we wish to locate either singly, or in groups as in the case of class failures. Then, every reference column is compared with every other column not in its failure class, and new distinguishability columns of ones and zeros are generated. There will be one column generated for every pair of columns of F that belong to different failure classes.

The resultant failure distinguishability table G, then, is analogous to the description of a multiterminal network, and we wish to minimize it such that all of the failure classes of G are distinguishable using the least number of input combinations. The methods of reduction used correspond to those used for finding a prime-implicant cover of a given switching function from its prime-implicant table.

When the inputs in the reduced fault-test table are applied to the unit under test, the particular machine we have will be found from the column which satisfies the set of test outputs.

Bounds on the Number of Tests

It is of interest to place upper and lower bounds on the number of tests required to identify the machine being tested. That is, how many columns there are in the reduced test matrix will determine the number of tests.

Obviously, the upper bound on the number of tests required would be equal to the number of failures in the original fault-test dictionary in that case where no minimization of the G matrix is possible. A lower bound on the required tests would be generated by the case where the fault-test matrix G contains a subset of rows which constitute a binary coding of all the columns. Reduction, in this case, would result in N tests,

$$N = 1 + \log_2 m ,$$

where m is the number of different fault classes.

Preparata (7) has shown the average of the minimal test schedule to be close to $2 \log_2 m$, which is not far from the minimum. Note that, when doing fault detection, the number of fault classes is one.

Serial Vs Fixed Test Schedules

Once the minimum test matrix has been derived, there remains a further consideration with regard to the test procedure. In what order, if any, should the individual tests be applied to the machine? If all the inputs in the test group are to be applied in every case, there is no need to worry about their order, since the entire sequence will be run anyway.

But, consider the desirability of examining the results of each step in a test sequence and letting the results of that step determine the next. It is possible to gain sufficient information to indicate some failure mode without the application of the entire test set. At any point in the test sequence, we need a way of measuring the information to be gained from applying any remaining available test input. From an information theory viewpoint, that test which most evenly partitions the test results is the optimum, provided all failure classes have the same probability of occurrence. For example, if each test result branches the failure possibilities into two groups (assuming a single binary output), we will want to choose the next test from a particular branch on the basis that such a test will most evenly divide the remaining possible failure classes into two groups. In a way similar to the case of the fault-test matrix, it is possible to reduce the number of levels in the test tree to L,

$$L = 1 + \log_2 m ,$$

where m is the number of failure classes.

The minimum stated above could be achieved where each test in the sequence divides the remaining failure classes into two equal parts.

TESTING SEQUENTIAL MACHINES

In sequential machines, tests are not independent. To place a network in a state in which a certain failure can be detected, a sequence of inputs must be applied. In other words, an incorrect state generated within the logic by a fault may require many test inputs before its effect is observable at the output. It is apparent then, that test sequences for sequential machines can become very long.

At this point, it might be well to review some of the assumptions underlying the automatic testing problem.

1. It is, of course, assumed in application that inputs and outputs of these machines are electrical and binary in nature. This does not, however, restrict the usefulness of the theory.

2. Further, to generalize the approach to the fullest extent, it is necessary to *not* restrict oneself to synchronous logic or to specialized configurations designed to succumb easily to certain test methods. It is expected, however, that the unit under test has been designed as a strongly connected machine even if it might not be so under failure.

3. It is expected that a complete description of the functioning design is available in the form of a logic diagram, state table, or other self-contained format amenable to application in computer simulation software.

4. Finally, as mentioned before, all expected modes of failure have been prespecified along with their effects on the system.

Philosophy of Diagnosing Sequential Machines

Diagnosis of a faulty system in broad terms entails discovering the set (sequence) of machine outputs which differ from those of the good circuit under a specified input set. Once these are found, one can relate them to one of the specified modes of failure.

The diagnostic task, as mentioned before, for a combinational digital machine is straightforward, assuming that, under failure, the combinational network does not behave sequentially. The set of input combinations required for complete testing never exceeds the number of failure modes and, in general, will be less.

Testing a sequential machine, however, requires the application of a set of input sequences. The length of these depends on the number of internal states of the machine, along with the number of its input and output terminals.

Testing methods have evolved along two paths. The first, or distinguishability philosophy, attempts to derive test sequences which distinguish between selected machines. By suitably intersecting these tests, it is possible to derive a set of input sequences which will isolate failures. The second, or black box, philosophy used by Moore (8) does not examine specific cases but interrogates the collection of machines, observing inputs and the corresponding outputs. Tests are then chosen on the basis of how well they partition the set of machines into equivalent failure classes.

State Table Methods

One way of checking a sequential machine is to verify that it operates in accordance with its design state table. Hennie (9) makes use of this method in a testing scheme which checks each state and the transitions from each state to every other state in the table. To accomplish this, the circuitry is first located in some known starting state. Then, an input chosen to take the circuit into some new state is applied and the result checked against the design state table. If all outputs occur correctly, then the circuit is good.

To place the circuit in a known starting state requires a synchronizing sequence which leaves the circuit in a known state. Since not all circuits have synchronizing sequences, one can use a "homing" sequence which, when applied, provides an output which uniquely defines the final state from which point a transition sequence will bring the circuit into the desired starting state. An input chosen to transfer the circuit to a new state is applied and the result checked by a distinguishing sequence. The distinguishing sequence provides a unique output sequence for each different starting state; thus, it identifies the initial state of the circuit to which it is applied. In applying the various sequences, one does not worry about whether the circuit is responding according to the state table, since an incorrect transition at any stage will appear under the application of the next distinguishing sequence, and the malfunction will be detected.

For a network having many states, this procedure will require large amounts of time and computer memory. The minimum number of distinguishing sequences that has to be applied is $m(n)$, where m is the number of inputs and n the number of states. In fact, Hennie states that the total number of test steps required can reach

$$m(n)(2L+2n-1)+n(n-1)+(n+1)L ,$$

where L is the length of the distinguishing sequence. Also, this method is not diagnostic in that it provides only the verification of correct operation.

A further disadvantage of this approach is that not all circuits have distinguishing sequences. In this case, a particular state might be identified by observing its response to two or more different inputs called characterizing sequences. The circuit must therefore be brought into a state as many times as characterizing sequences are needed to identify it. The resulting tests become many times longer.

Poage and McCluskey (10) expanded on this method by considering the state tables of the set of specified bad machines as well as of the good one. By combining all the state tables, an indication of the initial states for which distinguishing sequences cause different outputs for different machines is given. This procedure guarantees the shortest possible test sequences having diagnostic properties. Here again, the computer time and memory needed to handle the state tables is prohibitive.

Search Method

The basic search method was first utilized by Seshu and Freeman (11) and improved upon later by Seshu (12). Broadly speaking, it entails sequentially questioning a collection of machines and grouping them according to their responses until no further partitioning is possible under repeated questions. Each machine is treated as a black box and evaluated strictly on the basis of its input-output relationships.

The correctly operating network and the networks operating under the specified failures are all simulated. The entire set of machines is then placed in some starting state s_0. One of the criteria of this method is that there exist input combinations which uniquely set the states of any of the machines m_k. All input combinations are then applied to all the machines and the outputs recorded. One of these input combinations will be chosen to represent this step of the test sequence based on the output patterns of the group. The next step is a repetition of the first except that only those inputs which do not differ in more than *one* variable from the previously selected test input are allowed. This constraint follows from the philosophy of testing asynchronous networks in general.

The criteria by which the next inputs in the test sequence are chosen is closely related to the efficiency of the entire testing scheme. In Seshu's original method, a local search is used. That is, the suitability of a test for step n of the sequence is based upon the results of step $n - 1$ only. Global methods, which back k steps to the $n - k$ test, become prohibitively unwieldy with increasing k, although they could be applied for small k's. Chang (13) presents a well-organized treatment of distinguishability criteria by which test steps are chosen.

To appreciate the motivation behind the distinguishability criteria, let us review a few points.

In testing for failures we might want to ascertain one of three things. First: Is the unit under test operating correctly or not (is it machine m_0 or m_i)? Second: If the unit has failed, which m_i are we dealing with? Third: If the unit has failed, in which subgroup of failure classes does the unit fall? The last question properly arises when we are dealing with a network composed of smaller subassemblies such as logic cards or integrated circuits, and diagnosing a fault only means finding in which subassembly it is located so that it can be replaced. So, we can always divide the set of machines m_i into equivalence classes of failure. In case we wish to distinguish each and every failure, every machine falls into a separate class. Only in the case of fault detection do all the failed machines fall into only one class. In between, we assign machines to classes according to subassembly relationships.

Now, when questioning the set of machines, we would like responses to be alike for all machines in the same failure class. If this were possible, we would have a test which uniquely defines the failure mode in one step! However, our network, in general, does not have enough output combinations to partition every failure class and even so, it is highly unlikely that a single input test combination would provide such a fortunate result.

Seshu's work utilizes an *information gain* criteria, which measures the ability of a test to partition each fault into a separate channel. For a test which generates m partitions, its information gain can be computed as follows: Referring to Fig. 3, let $(i+1,\ j_1)$, $(i+1,\ j_2), \ldots, (i+1,\ j_m)$ be the m equivalence classes which result by applying test T_k to (i,j)—the equivalence class at the ith test level and the jth partition. Then the information gain for test T_k is

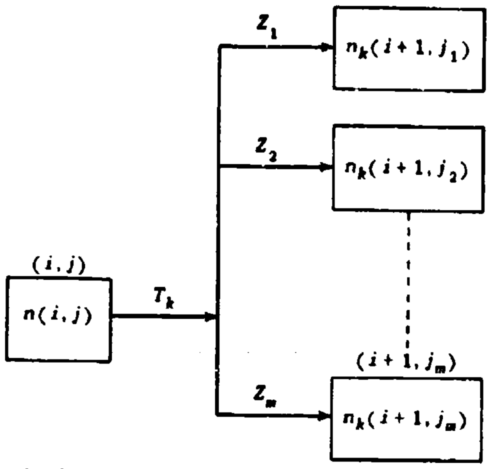
$$\beta_k = -\sum_{s=1}^{m} \frac{n_k(i+1,\ j_s)}{n_k(i,\ j)} \log_2 \frac{n_k(i+1,\ j_s)}{n_k(i,\ j)}, \tag{1}$$

where all faults in (i,j) are assumed equiprobable.

The information gain of Eq. (1) is suitable for binary partitioning. It can be extended to the m-partitioned case by taking the summation of the gains of $m-1$ binary partitions. If a priori probabilities are assigned to failures, suitable weightings can be applied to the terms of the summation.

When a test level T_k on equivalence class (i,j) gives no further partitioning, there is an information gain of zero, a new reset condition is tried, and the process continues. If, after all resets have been tried, no further partitioning is possible, then the machines in that branch of the test tree are indistinguishable by this method, and the process proceeds with other classes until all possible partitioning is completed. The result is a test tree which can be applied to the network in question, branching as indicated by the output Z_m until a path end is reached indicating what failure or group of failures applies.

In his paper, Chang (13) gives a better method of measuring information gain where failures in a submodule are grouped into the same equivalence class. The essential idea is that one would like to apply a test which would distinguish every machine m_j from every other machine m_k not in its equivalence class. If every machine *is* in its own class, there are $N(N+1)/2$ pairs to be distinguished. A test is most useful, then, if it distinguishes the greatest number of pairs of machines. Thus, we count the pairs formed by each machine in $(i+1,\ j_k)$ and those in $(i+1,\ j_l)$ as given in Eq. (2):



<u>Notation</u>

(i, j) = Equivalence class at ith level and jth partition

$n(i, j)$ = Number of machines in (i, j)

$n_k(i+1, j_s)$ = Number of machines in $(i+1, j_s)$ resulting from applying T_k to (i, j)

T_k = Test at level k of sequence

Z_m = Output resulting from test T_k

Fig. 3 - Partitioning by test step T_k

$$\begin{aligned}\gamma_k = {} & n_k(i+1,\ j_1)\ [n_k(i+1,\ j_2) + n_k(i+1,\ j_3) + \cdots + n_k(i+1,\ j_m)] \\ & + n_k(i+1,\ j_2)\ [n_k(i+1,\ j_3) + \cdots + n_k(i+1,\ j_m)] \\ & + \cdots,\ + n_k(i+1,\ j_{m-1})\ [n_k(i+1,\ j_m)] \\ = {} & \sum_{s=1}^{m-1} \left[n_k(i+1,\ j_s) \cdot \sum_{t=s+1}^{m} n_k(i+1,\ j_t) \right] . \qquad (2)\end{aligned}$$

Now, when we extend this to fault location within modules, we redefine γ_k by removing from the count those fault pairs that are distinguished by T_k but are associated with the same module.

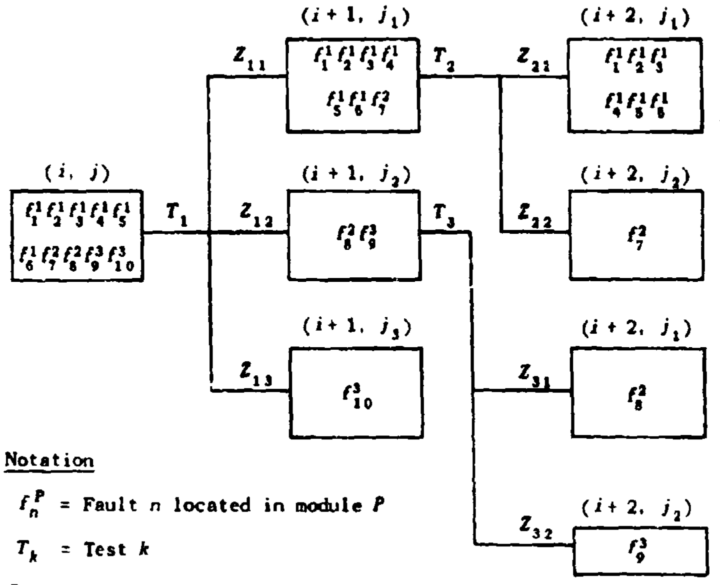
Let $n(i+1,\ j_m)_P$ denote the number of faults in $(i+1,\ j_m)$ associated with module P. Thus, by the same reasoning we used to obtain Eq. (2), we get

$$\sum_{s=1}^{m-1} \left\{ \sum_{P} \left[n_k(i+1,\ j_s)_P \sum_{t=s+1}^{m} n_k(i+1,\ j_t)_P \right] \right\}$$

fault pairs that need not be distinguished. Subtracting this from γ_k leads us to our measure λ_k:

$$\lambda_k = \sum_{s=1}^{m-1} \left\{ \sum_{P} \left[n_k(i+1,\ j_s)_P \sum_{t=s+1}^{m} n_k(i+1,\ j_t) - n_k(i+1,\ j_t)_P \right] \right\}. \qquad (3)$$

For an example of a test tree generated by this procedure, see Fig. 4.



Notation

f_n^P = Fault n located in module P

T_k = Test k

Z_{kj} = Output for equivalence class j resulting from test k.

Fig. 4 - Partitioning of faults to within modules

Chang's procedure has the advantage of generating shorter test sequences when testing to within modules is applicable, and it eliminates the $\log_2$ calculations, thus effecting savings in computer time.

Jones and Mays (14) have experimented with a fault-detection system based on Seshu's local search method but incorporating changes that improve the ability of the program to find test sequences for all listed faults. Instead of abandoning a test path when output patterns no longer partition the equivalence class, the state of the internal feedback lines are examined, along with the outputs. Information gain is then figured on the response of the memory states as well as on the output. If improvement is obtained by this procedure, the test sequence is continued until either partitioning based on output states or a set limit on the number of steps is reached.

The local search method, while it discovers faults rapidly in its early stages, tends to wander and be less productive when few faults remain unpartitioned. Jones and Mays therefore cause the program to break into a new random search phase when local search is sensed to have proceeded to unproductivity. The first task of this phase is to try all reset states for any new output partitioning. Then, if undetected faults still remain, a reset combination is chosen as a starting point and an input combination chosen at random is applied while looking for a new state defined by the levels of the feedback and output lines. Each new state that detects a fault is recorded, and the process is repeated until all faults or all states have been exhausted.

When the states that detect the remaining faults have been ascertained, a search is made to find the minimum transfer sequences between the reset states and closest states detecting faults. These last sequences are added to the local search phase sequences, thus making up the total test sequence.

SUMMARY

Systematic procedures for the diagnosis of combinatorial networks have been devised. In the case of sequential networks, nothing found in the literature offers complete testing with reasonable computer costs.

If tests exist for the determination of given failures in a combinational circuit, then the critical path or sensitivity methods will find them. Circuitry known to be combinatorial would be best tested by methods other than the general sequential approach.

Two approaches covering the testing of sequential networks are the state table and the directed search methods. Prohibitive computer time and memory requirements bar the use of state table methods as they now stand. The directed search method, while it loses efficiency in the last stages of test generation, is (on the average) extremely powerful.

It appears from the latest efforts that approaches combining variations on the directed search method and, perhaps state table methods, will bear the choicest fruit.

FUTURE WORK

To assemble a diagnostic system for digital networks, three things are required:

1. A programmable logic simulator,
2. A diagnostic test generating system, and
3. A test-sequence application and control unit.

1. A synthesis of software simulators suitable for sequential logic is no small task by itself. It must, for diagnostic applications, be flexible enough to handle programmable fault conditions and be able to recognize oscillations and races under control of the diagnostic test generator. Considering the complexity of this portion of the system requirements, it would be advantageous to utilize existing software, making whatever modifications are necessary.

2. The literature (15,16), indicates that directed search test generators appear to promise the greatest flexibility, the most testing power, and the most economical use of computer facilities.

A programming system written by Seshu and incorporating a logic simulator and diagnostic test generator has been converted to CDC-3600 use by Chang and Goetz at Bell Telephone Laboratories. Arrangements have been initiated for obtaining a copy of this program to use as an experimental tool. It is an improved version of Seshu's original local search routine with strategies for making random and combinational test searches at program discretion. The program would provide a vehicle for experimenting with different approaches. Replacing Seshu's information gain criteria with Chang's distinguishability criteria for testing within modules should realize a reduction in test sequence sizes.

Applying the Jones and Mays integrated fault-detection method to a system with fault distinguishability capabilities might result in an efficient diagnostic system with more complete test-finding capabilities.

Also, a Fortran IV logic simulator is available from the University of Buffalo, and arrangements have been made to obtain it.

3. Once a test sequence has been generated, it is necessary to have a device which will apply the sequence to the network in question and properly interpret the results. It is impractical to assign the job to the computer on which the diagnostic sequences are generated. Rather, it is anticipated that test sequences written on magnetic tape would be applied by a special unit consisting of either a digital tape deck with special logic or a small general purpose computer. Suitable interface flexibility would have to be provided for handling the voltage, impedance, and timing requirements of various circuitry.

REFERENCES

1. Kautz, W.H., "Fault Testing and Diagnosis in Combinational Digital Circuits," IEEE Trans. Computers C-17:352 (Apr. 1968)

2. Hornbuckle, A., and Spann, R., "Diagnosis of Single Gate Failures in Combinational Circuits," IEEE Report R-68-158, June 1968

3. Sellers, F.F., Jr., Hsiao, M.Y., and Bearnson, L.W., "Analyzing Errors with the Boolean Difference," IEEE Trans. Computers C-17 (No. 7):676 (July 1968)

4. Amar, V., and Condulmari, N., "Diagnosis of Large Combinational Networks," IEEE Trans. Electronic Computers (Correspondence) EC-16:675 (Oct. 1967)

5. Roth, J.P., "Diagnosis of Automata Failures: A Calculus and a Method," IBM J. Res. Dev. 10:278 (July 1966)

6. Galey, J.M., Norby, R.E., and Roth, J.P., "Techniques for the Diagnosis of Switching Circuit Failures," IEEE Trans. Commun. Electron. 83 (No. 74):509-514 (Sept. 1964)

7. Preparata, F.P., "An Estimate of the Length of Diagnostic Tests," Third Annual Princeton Conference on Information Sciences and Systems, Mar. 1969 (to be published)

8. Moore, E.F., "Gedanken-Experiments on Sequential Machines," in "Automata Studies," Princeton Annals of Mathematics Studies No. 34, Princeton:Princeton University Press, 1956, pp. 129-153

9. Hennie, F.C., "Fault Detection Experiments for Sequential Circuits," Proc. Fifth Annual Switching Theory and Logical Design Symposium, Princeton:Princeton Univ., 1964

10. Poage, J.F., and McCluskey, E.J., "Deviations of Optional Test Sequences for Sequential Machines," Proc. Fifth Annual Symposium on Switching Theory and Logical Design, Princeton:Princeton Univ., Nov. 1964, pp. 121-132

11. Seshu, S., and Freeman, D.N., "The Diagnosis of Asynchronous Sequential Switching Systems," IRE Trans. Electronic Computers EC-11:459-465 (Aug. 1962)

12. Seshu, S., "On an Improved Diagnosis Program," IEEE Trans. Electronic Computers EC-14:76-79 (Feb. 1965)

13. Chang, H.Y., "A Distinguishability Criterion for Selecting Efficient Diagnostic Tests," 1968 Spring Joint Computer Conference, Washington, D.C.:AFIPS, Vol. 32, pp. 529-534

14. Jones, E.R., and Mays, C.H., "Automatic Test Generation Methods for Large Scale Integrated Logic," IEEE J. Solid-State Circuits SC-2 (No. 4):221-226 (Dec. 1967)

15. Manning, E.G., and Chang, H.Y., "A Comparison of Fault Simulation Methods for Digital Systems," Digest of the First Annual IEEE Computer Conference, Chicago: 1967, pp. 10-13

16. Breuer, M A., "Hardware Fault Detection," 1968 Fall Joint Computer Conference, San Francisco:AFIPS, Vol. 33, pp. 1502-1503

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Naval Research Laboratory Washington, D.C. 20390	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE

DIGITAL DIAGNOSTIC TECHNIQUES: SURVEY AND RECOMMENDATIONS

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
An interim report on a continuing program.

5. AUTHOR(S) (First name, middle initial, last name)

William R. Smith

6. REPORT DATE October 2, 1969	7a. TOTAL NO. OF PAGES 18	7b. NO. OF REFS 16

8a. CONTRACT OR GRANT NO. NRL Problem B01-04	9a. ORIGINATOR'S REPORT NUMBER(S) NRL Report 6938
b. PROJECT NO. Project RR003-02-41-6150	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Department of the Navy (Office of Naval Research), Washington, D.C. 20360

13. ABSTRACT

Finding efficient testing procedures for digital machines has become of prime concern to the industry due to high circuit densities brought about by evolution in semiconductor device technology.

The number of inputs required for testing can be reduced from the set of all possible input combinations to a smaller group derived from a dictionary of tests which defines unique relationships between failure mode and output response. Reduction of the test dictionary by prime-implicant minimization methods and by serial application of the reduced set can lead to manageable test programs. These involve sequence lengths characterized by a lower limit, bounded by the logarithm of the number of initial table entries.

Testing sequential machines is especially cumbersome due to the dependency of circuit response on previous input sequences. Lines of attack include checking for consistency between circuit response and design state table. A more manageable scheme involves grouping machines of specific failures according to their responses to selected test inputs. A proper choice of inputs for each test step and the adaptive application of the resulting sequences provides relatively efficient fault-location programs.

Practical schemes for complete testing have yet to be found. Adaptive sequence generators with programmed options for state-table and combinational analysis appear to yield more promising results.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6801

Security Classification

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Digital computer fault location programs Diagnostic routines Testing for computer failure Test-generating algorithms for fault location						